Proceedings of the 2005 Winter Simulation Conference
M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, eds.

A FRAMEWORK FOR FAULT-TOLERANCE IN HLA-BASED DISTRIBUTED SIMULATIONS

Martin Eklöf
Farshad Moradi

Swedish Defence Research Agency (FOI)
Dept. of Systems Modeling
SE-172 90 Stockholm
SWEDEN

Rassul Ayani

Royal Institute of Technology (KTH)
Dept. of Microelectronics and Information Technology
SE-164 40 Stockholm
SWEDEN

ABSTRACT

The widespread use of simulation in future military systems depends, among others, on the degree of reuse and availability of simulation models. Simulation support in such systems must also cope with failure in software or hardware. Research in fault-tolerant distributed simulation, especially in the context of the High Level Architecture (HLA), has been quite sparse. Nor does the HLA standard itself cover fault-tolerance extensively. This paper describes a framework, named Distributed Resource Management System (DRMS), for robust execution of federations. The implementation of the framework is based on Web Services and Semantic Web technology, and provides fundamental services and a consistent mechanism for description of resources managed by the environment. To evaluate the proposed framework, a federation has been developed that utilizes time-warp mechanism for synchronization. In this paper, we describe our approach to fault tolerance and give an example to illustrate how DRMS behaves when it faces faulty federates.

1 INTRODUCTION

Simulation models are increasingly being used as integral parts of modern military command and control and decision support systems. The nature of many of today's simulation models, in terms of processing capacity required for execution or decomposition to promote reuse and/or connection of geographically dispersed units, shows the importance of methodology for distributed simulation. In this context the High Level Architecture (HLA) is a widely used standard for distributed simulations. In HLA, a simulation is referred to as federation, whereas an individual simulation component is referred to as a federate. The decomposition of a simulation system certainly has its merits, but will typically lead to a higher failure rate (Kiesling 2003). In the perspective of a military decision support system the failure of a critical simulation component is often unacceptable, especially when time is a constraining factor. Thus, support for fault-tolerant distributed simulation is crucial in such systems. Thus, we need some mechanisms for detecting errors in the simulation execution, as well as measures for restoring an erroneous federation execution.

The HLA provides basic functionality for restoring an unsuccessful simulation execution, through the save and restore features of the federation management services. However, no means of detecting an error, or automatically restoring an erroneous simulation execution, are defined by the HLA. Also, the save and restore facilities are used in the local scope of a federate, meaning that a saved state is not automatically distributed outside the node where the federate resides. To cope with an unreliable host environment of a federate, in terms of hardware crashes or lost network connections, it is necessary to enable state saving at a "global" level and resumption of a federate execution in a new host environment. These functions are the fundament for what is usually referred to as federate migration, i.e. the transfer of federates between different host environments.

In our previous work we explored the possibilities for migration of federates using the peer-to-peer-based Distributed Resource Management System (Eklöf, Sparf, and Moradi 2004). However, in our previous work the decision to migrate a federate was based on the willingness of workstation owners to share their computing capacity for simulation execution. Thus, the system did not consider detection of a failure and migrations of federates were based upon user requests.

In this paper we present a revised architecture of the DRMS and outline a partial implementation of the concept, based on Web Services. More specifically, this paper will address a mechanism for fault-tolerance in time-warp based federations. The fault-tolerance mechanism does not consider software errors, in terms of a simulation model producing erroneous result, but handles cases where the host environment of a federate crashes, the federate itself crashes for some reason or the federate's link to the RTI is lost. Moreover, we assume that the federates executed

within the scope of DRMS are transferable, meaning that they are not bound to a specific piece of hardware and can easily be migrated between different host environments.

2 BACKGROUND

As computers in a distributed simulation do not share a common clock it is required that a virtual time, usually referred to as logical time, is introduced for each member of the simulation. A time synchronization protocol is used to maintain the logical time of members and ensures the causal ordering of events.

2.1 Optimistic Synchronization in HLA

The time-warp approach to synchronization, proposed by Jefferson (1985), is the most well known optimistic synchronization protocol. In the time-warp protocol, logical processes (LPs) are allowed to process events optimistically, which means that events may arrive that have a smaller time-stamp than previously processed events. This implies that LPs are also permitted to send messages optimistically, which means that sent messages could later be cancelled. The cancellation is performed by sending anti-messages to the receivers of the original events. An inevitable aspect of the time-warp protocol is the ability of an individual LP to restore to a previous state in its past. This process is referred to as rollback. Rollback is triggered if an LP receives a message in its past, or if a processed event is annihilated by an anti-message.

The time-warp protocol has also been utilized in HLA-based distributed simulation. The bulk of research in this area addresses development of middleware that will shield the developer of a federation from the often complex task of implementing time-warp. In (Wang et al. 2004) the issue of time-warp is investigated in the context of integrating COTS simulation packages and the HLA. A middleware for management of the rollback mechanism is presented and evaluated. In (Yan, Sun, and Zhong 2003) a time management meta-level is introduced between the RTI and individual federates. This layer uses a computational reflection technique to free the developer of an optimistic federate from the complex task of implementing the rollback mechanism. Huang et al. (2003) describes the addition of a middle layer, referred to as the Smart Time Management (STM), which aims at unifying various time management schemes, such as time-stepped, event-driven and optimistic time advancement approaches in the HLA. In (Vardanega and Maziero 2001) a generic rollback manager for optimistic HLA simulations, based on computational reflection techniques, is presented. The manager implements state saving and manages rollback for optimistic federates.

2.2 Fault-Tolerance in HLA

A distributed simulation, or distributed system for that matter, has a higher failure rate than a simulation, or system, executed on a single machine. However, a failure in a distributed simulation is often partial, that is, one of the components of the system fails. The failure may, or may not, affect other components of the system. In the past, several techniques for fault-tolerance in distributed systems have been developed. These techniques can be classified into two main categories; replication based and check-pointing based (Damani and Garg 1998). In replication based approaches one or more copies of an LP is maintained in addition to the main LP. In case of failure, one of these replicas will take the failed LP's place. In check-pointing based approaches, states of the individual LPs are saved on stable storage. In case of failure, an LP is restarted using the last stable state saved on stable storage.

According to Kiesling (2003) research in fault-tolerant distributed simulation has been quite sparse. Work in application of fault-tolerance techniques in the context of HLA is even more abundant. However, there is some work that aims in this direction. Lüthi and Berchtold (2000) provide a structured view of fault-tolerance in parallel and distributed simulations and possible solutions are presented. In (Lüthi and Großmann 2001) a Resource Sharing System (RSS) is presented that in a future extension could serve as the basis for fault-detection, check-pointing and replication of federates. In (Berchtold and Hezel 2001) a concept, named R-FED (Replica Federate), in support of fault-tolerant HLA federations is presented. As the name implies, the approach is based on replication of individual federates in a federation. Several papers address the issue of federate migration, which is an important cornerstone in designing an infrastructure for fault-tolerant distributed simulation, see for example (Eklöf, Sparf, and Moradi 2004; Tan, Persson, and Ayani 2004; Bononi, D'Angelo, and Donatiello 2003; Cai, Turner, and Zhao 2002; Lüthi and Großmann 2001). However, these papers usually address federate migration in the context of load-balancing and do not explicitly address fault-tolerance.

The present version of HLA is IEEE 1516-2000. Currently, work is carried out to define the next version of HLA, through the HLA Evolved (Möller, Karlsson, and Löfstrand 2005). By the end of 2005, or early 2006, this work is expected to be complete. An interesting aspect of HLA Evolved is that fault-tolerance has been given more focus than before. HLA Evolved will provide a common semantics for failure and mechanisms for fault-detection. At the core, two additions have been made to the Management Object Model (MOM), namely *federate lost* and *disconnected.* These interactions provide the basic mechanisms for signaling a fault from the context of a federation, through *federate lost*, and from the perspective of a federate, through *disconnected*. Upon failure, the RTI has the

responsibility to do resign on behalf of the lost federate using the *Automatic Resign Directive*. This line of development is important for future realization of fault-tolerant distributed simulations, based on the HLA.

3 DISTRIBUTED RESOURCE MANAGEMENT SYSTEM – DRMS

In the following section the DRMS is presented in the context of network-based M&S. Next, the mechanism for fault-tolerance implemented in DRMS, to support robust execution of time-warp based federations, is described.

3.1 Network-Based M&S

The DRMS provides computing capacity for reliable execution of simulations and is an essential part of a network-based modeling and simulation environment, referred to as NetSim, being developed at the Swedish Defense Research Agency (Eklöf, Ulriksson, and Moradi 2003). NetSim supports collaborative simulation development and execution within and between organizations and will thus promote increased use and reuse of simulation models and also lead to increased quality of work in the M&S development process.

Figure 1 presents an overview of the service-oriented architecture of NetSim. The uppermost layer comprises various NetSim tools, dedicated to M&S-related tasks, for instance tools for composition of federations by a single user, or collaborative development of federations by a number of users. The NetSim tools derive their functionality from NetSim specific services, denoted DRMS, CC and Repository in Figure 1. As stated above the DRMS provides computing capacity for reliable execution of simulations. The CC (Collaboration Core) provides services for collaborative work, whereas the Repository provides services for look-up of available resources on a network. The NetSim specific services are based on various overlay network service technologies, such as Web Services, Grid Services and the HLA RTI. These are just examples of network technologies that could be deployed to achieve the goals of the NetSim environment. Throughout all layers in Figure 1, a common syntax and semantics for description of resources is used to promote interoperability. Moreover, security is considered an integral part of all layers.

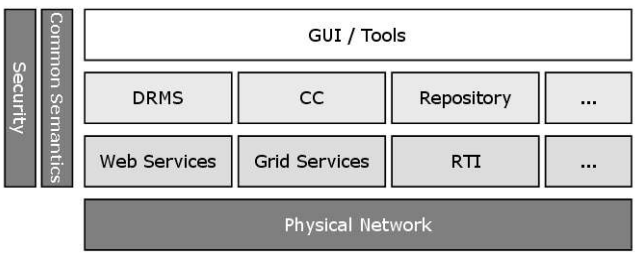

Figure 1: Architecture of NetSim.

3.2 DRMS Concept

DRMS comprises two basic service types, namely a *worker service* and a *coordinator service*. A worker is responsible for execution of one or more jobs, whereas a coordinator is responsible for the coordination of one or more workers in managing a batch of jobs. In addition to these basic services, the DRMS is dependant on a *repository service*. A repository is used by a worker to advertise its presence on the network and also its availability for execution of various jobs. Furthermore, the repository is used by a coordinator for localization of available workers. A repository also contains advertisements of other resources available on the network and is therefore used as entry point when worker services fetch resource files and executable code.

3.3 Fault-Tolerance Approach

The main idea of our approach to fault-tolerance in time-warp based federations is to use a check-pointing mechanism to enable restoration of a federation upon failure. The check-pointing is done by means of the RTI communication infrastructure, utilizing an extension to the Federation Object Model (FOM). In this context, a checkpoint (CP) represents the state of a federate at a specific point in time, for example through a vector of state variables. The check-points are saved in a stable storage component, which is also a member of the federation execution. An important feature of the check-pointing is to make sure that the individual state represents a federate at a point in time that could not suffer from rollback. This means that the federate must report checkpoints to stable storage, which represent the state of the federate at a point in time that is less than the smallest timestamp of a message that could ever be delivered to the federate. In this way, it will always be safe to use the check-point for restoration. The state-saving is not synchronized throughout the federation, but federates report their states to stable storage individually. The mechanism for check-pointing is illustrated in Figure 2.

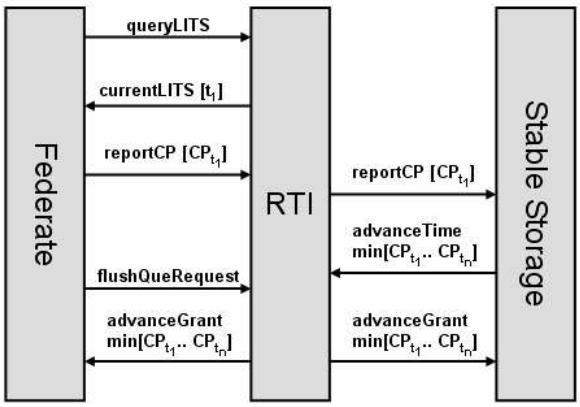

Figure 2: Check-Pointing Mechanism in the DRMS.

First, the concerned federate uses the queryLITS (Least Incoming Time-Stamp) method of its RTI ambassador to extract the timestamp of the next TSO (Time-Stamp Order) message that it may have to process. The federate uses this value to produce a checkpoint that could not be invalidated in the future. The checkpoint can not be cancelled since the federate can never be roll-backed prior to this time and thus, it represents a safe state of the federate. The LITS is used as timestamp when reporting the checkpoint to the stable storage. When the stable storage receives a checkpoint, it calculates the minimum timestamp of the checkpoints that have most recently been reported from each federate in the federation. This minimum timestamp is then used by the stable storage for requesting advancement of time. Upon requesting flush of the RTI queues, the individual federates will receive time advancement grants based on the timestamps of the supplied checkpoints. The granted time represents the GVT (Global Virtual Time) of the federation. Note that this time does not represent the actual (local) time of a federate. GVT is the boundary up to which the simulation execution is regarded as complete by all participants and is used to perform garbage collection of saved states to free memory space.

The purpose of allowing the stable storage to control advancement of GVT is crucial for migration purposes. During the migration of a federate the GVT must not be advanced beyond the time-stamp of the checkpoint that the migrating federate will rely upon for its restoration. The mechanism described above will make sure that this will not occur.

3.4 Migration of Federates

Below, the process of migrating a federate upon failure is described. When an individual federate is deployed in a new host environment, the startup scheme differs slightly from the normal case. This process is illustrated in Figure 3.

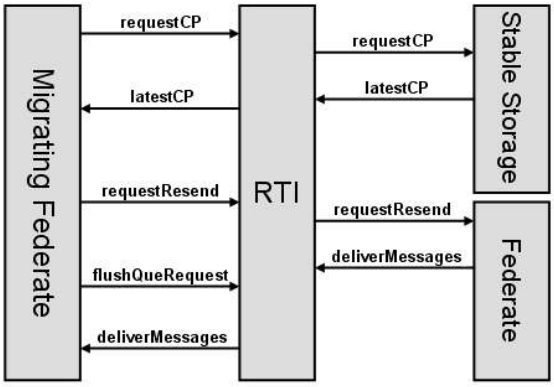

Figure 3: Federate Migration Process.

Initially, the federate requests the most up-to-date checkpoint of its state from stable storage. The federate restores its state based on this checkpoint. Then the federate makes a request to all participants to resend all messages whose timestamp is greater than GVT. Federates that have produced messages to the concerned federate, resend these messages. When the migrated federate requests flush of the RTI queues, it will receive the missing messages and can then resume the execution.

The described mechanism requires additional customization of participating federates and introduction of four supplementary interactions to the FOM, namely reportCP, requestCP, latestCP and requestResend. Extra interactions required by the fault-tolerance mechanism are outlined in Table 1.

Table 1: Interactions Added to the FOM to Support the Fault-Tolerance Mechanism (P = Publish, S = Subscribe).

Interaction	Description	Federate	Stable Storage
reportCP	Reports checkpoint to Stable Storage	P	S
requestCP	Requests latest checkpoint from stable storage	P	S
latestCP	Delivers latest checkpoint to migrated federate	S	P
requestResend	Requests resend of messages from GVT	P, S	-

3.5 Services and Ontology

The DRMS concept presented in section 3.2 has been partially implemented. The implementation is based on Web Services, the Axis platform (Saleem 2004), and Semantic Web technology, through use of the Jena toolkit (McBride 2002). In the following section the implementation is described briefly. The following components of the implementation are described:

- DRMS ontology
- RemoteJobService
- ResourceRepositoryService
- ExecutionService.

To enable uniform and semantically rich descriptions of resources within the environment, a *DRMS ontology* is used. In near future, this ontology will be aligned with a general NetSim ontology that is currently under development. The DRMS ontology comprises constructs for description of simulation models and computing resources.

The main purpose of the ontology is to promote a shared view of information throughout the environment and facilitate localization and matching of resources. The chosen language for its representation is the Web Ontology Language (OWL) (McGuniess and van Harmelen 2004). The expressiveness of OWL is sufficient for representation of information required by the DRMS. The language also enables inference over information, which is used to match resources in the implementation.

The *RemoteJobService* implements a worker as described in section 3.2. When deployed on a workstation, this service announces its presence on the network by registering an announcement in a repository. The announcement is represented by a meta-model, defining the features of the RemoteJobService's host environment. This includes aspects such as the workstation's hardware configuration, OS type and version etc. The meta-model is an instance based on the DRMS ontology. Table 2 outlines the service interface of the RemoteJobService.

The *ResourceRepositoryService* is a simple implementation of a repository as described in section 3.2. This service supports storage of meta-models, such as the meta-model describing the RemoteJobService's host environment. The interface of the ResourceRepositoryService includes methods for registering, deletion and lookup of meta-models. The lookup can either respond with the entire content of the ResourceRepositoryService, or a subset of registered meta-models, defined by a search query. Table 2 outlines the service interface of the ResourceRepositoryService.

The *ExecutionService* is an implementation of a coordinator as described in section 3.2. An ExecutionService is utilized by the NetSim environment when a single user, or group, requests execution of a scenario (federation). The main tasks of the ExecutionService are to automatically setup a federation and to monitor the federation execution. Table 2 outlines the service interface of the ExecutionService. Figure 4 gives a schematic view of the interrelation of DRMS services.

Table 2: Service Interfaces of the DRMS Implementation.

Service	Method
RemoteJobService	allocateJob(*Meta-model*)
	startJob(*Id*)
	stopJob(*Id*)
	getJobStatus(*Id*)
ResourceRepositoryService	addModel(*Meta-model*)
	deleteModel(*Meta-model*)
	getModels()
	getSubset(*Query*)
ExecutionService	requestExecution(*Scenario*)
	finalizeExecution(*Id*)
	getScenarioStatus(*Id*)

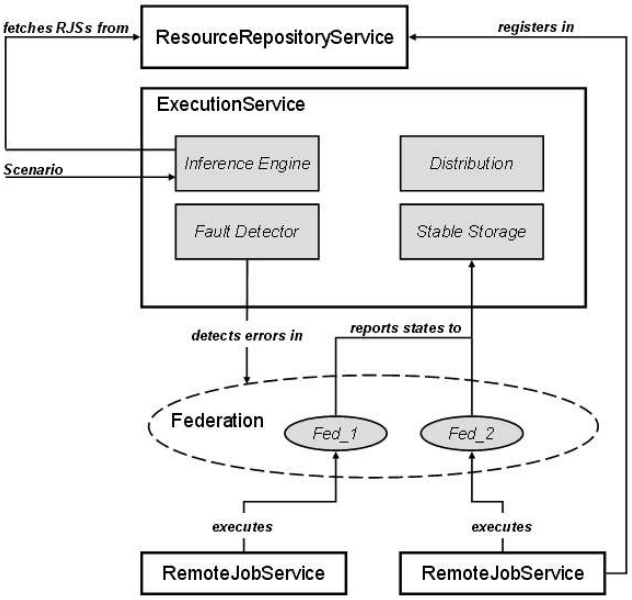

Figure 4: Interrelation of DRMS Services.

When the NetSim environment requests execution of a federation, it feeds the ExecutionService with a scenario description. The scenario description comprises meta-models for all federates that are part of the federation. In order to distribute the federates in the federation, to suitable nodes in the network, the ExecutionService fetches meta-models, representing RemoteJobServices, from the ResourceRepositoryService. Next, the ExecutionService determines a suitable distribution of federates, by matching meta-models of the federates with meta-models of available RemoteJobServices. The matching procedure utilizes an inference engine and a set of pre-defined rules to find a suitable distribution of federates over available RemoteJobService nodes. If the allocation of one or more federates is accepted by a RemoteJobService, it starts downloading the required executable code and possible resource files. The URLs to these files are defined in the meta-models representing the federates in question. When the download process is completed, the ExecutionService signals start-up of the federation to concerned RemoteJobServices.

To enable fault-tolerant execution of the federation the ExecutionService comprises a stable storage and a fault detector component. These components are members of concerned federation through a common federate. The stable storage stores checkpoints reported from federates in the federation, whereas the fault detector detects failed federates in the federation and initiates preventive measures to resolve these errors. The error detector detects the failure of a federate by means of the HLAfederate object of the MOM, which is deleted if the link to the RTI is broken. As an additional measure the error detector calculates the time passed from the last reported checkpoint and if this value

exceeds a pre-defined time, the federate is not longer considered active. When a federate crashes, or its network connection is simply lost, the fault-detector initiates redistribution of the lost component in the inference engine. The inference engine finds a new host environment for the federate under consideration, given the requirements of the federate as defined by its meta-model, and allocates the job to the RemoteJobService node.

3.6 An Example

Below, we look at an example to illustrate how the DRMS handles the occurrence of fault in a federation. In order to test the proposed fault-tolerance approach, a simple time-warp federation has been developed. This federation consists of four federates, which form a fully connected network, i.e. each federate is able to send messages to all other federates. In the test federation, processing of a message simply means updating a statistics object that describes the message exchange during a federation execution. Each federate randomly schedules events. This means that at random points in time, a federate sends a message to a randomly selected joined federate. The federates are initiated using disparate random seeds, causing the event scheduling to be based on different random streams within each federate. The federates process and produce events optimistically, thus when a message is received in a federate's past, a rollback is triggered. Similarly, when an anti-message is received that will annihilate an already processed message, a rollback is also triggered. The rollback mechanism relies on a record of locally saved checkpoints. Advancement of GVT is used to garbage collect the record.

Consider a federation comprising four federates, labeled A, B, C and D. Table 3 describes the state of the federates, in terms of their message queues, as GVT equals 10. Grey cells represent messages that have been processed by the federates, whereas the white cells represent unprocessed messages.

The process of federate migration, in case of failure, resembles a rollback to GVT. However, in this case special attention is required since the action is not coordinated. In an ordinary rollback the concerned federates send anti-messages to annihilate invalid messages. In case of failure this is not possible and must be handled separately be each federate. For instance, consider the case when federate A in Table 3 crashes. When federate A is absent, federates B, C and D continues their execution. However, since no checkpoints are reported from federate A, the stable storage will not request time advancement greater than 10. When the fault detector has detected the lost federate, and a new host environment has been identified by the inference engine, the failed federate is deployed at a new node. Next, the migrated federate fetches the latest saved state in stable storage, as defined in section 3.4. When the federate has restored using the state from stable storage, it requests resend of messages. This request also means that the non-migrated federates must annihilate messages received from federate A. In this case federate C must annihilate message A15 and federate D message A17. This will trigger retraction of message C21 in federate B, but no rollback will be initiated since the message has not been processed yet.

When the potential message annihilation is finalized federate B, C and D resend messages destined for federate A, whose time-stamp is greater than GVT. In this case, given that the queue configurations do not change during migration, federate B resends message A12 and A14. Next, federate A reschedules the received events and the federation resumes normal execution.

Table 3: Message Queues in Federates of Test Federation when GVT Equals 10; Grey Cells Represent Processed Messages, whereas White Cells Represent Unprocessed Messages.

Federate	IN	OUT
A	B12	C15
	B14	D17
B	D9	A12
	C12	A14
	C21	-
C	D11	B12
	A15	B21
D	A8	C11
	A17	_-

4 DISCUSSION

As modeling and simulation is integrated in various environments and used as a tool in the decision process, the requirements on the supporting infrastructure will be high. An important aspect in this is to enable fault-tolerant distributed simulation, since this ensures a robust execution environment that can respond to user needs in a timely fashion. The de facto standard for distributed simulation, the HLA, which is widely used throughout the military domain, does not treat fault-tolerance extensively. Nor has this topic been treated by the research community comprehensively. Given this, it is crucial to develop efficient and scalable methods for fault-tolerance in HLA-based distributed simulation.

Our approach is based on implementing fault-tolerance mechanisms within the framework of the HLA, i.e. communication related to the fault-tolerance mechanism is sent over the RTI. This of course implies that individual federates conform to the requirements imposed by the fault-

tolerance mechanism, in terms of publishing and subscribing to the interactions defined in Table 1. Currently, these aspects must be implemented by each federate individually. In the long run, it is desirable to implement these features in a generic fashion, through some kind of middleware system, to simplify the deployment of federates within the DRMS.

Introducing fault-tolerance mechanisms in M&S infrastructures will impose a cost. Regardless of the approach taken, replication based or check pointing based fault tolerance, the infrastructure must cope with increased network traffic and consumption of more hardware resources. Thus, it is important to evaluate the cost of having fault-tolerant simulations to determine when the approach is beneficial. Furthermore, aspects of fault-tolerance should be considered in the early phases of the FEDEP process. For instance, it is important to determine what levels of fault-tolerance are required by different components of the simulation in the context of what degree of degradation is acceptable for a given target (Möller, Karlsson, and Löfstrand 2005).

The proposition for a next version of the HLA standard, the HLA Evolved, will simplify the process of developing fault-tolerant federations. In this standard, a common semantics for failure and mechanisms for fault detection are provided. Still, there are others issues to resolve as well. Given the failure of a critical component in a federation, whose original host environment is not accessible for its restoration, a mechanism for deployment of the component in a new host environment is required. This can be solved through replication of the component, or through utilization of check-pointing, at a global level. Our work shows that it is feasible to use a check-pointing based scheme, employing the RTI communication infrastructure, to enable fault-tolerance in time-warp federations. However, it should be noted that this kind of check-pointing is tightly coupled with the time synchronization protocol of the federation. Other, or complementary, solutions have to be provided for other synchronizations protocols, or cases with mixed synchronization protocols. Also, the test federation used in this work comprises no complex issues of ownership of objects in the federation. In more complex federation types, the issue of transferring ownership of objects between federates, in case of failure, must be resolved as well.

5 FUTURE WORK

In estimating the cost of fault-tolerant distributed simulation, based on our approach, it is of interest to look at the size of the check-points reported to stable storage and how this potentially will degrade the simulation execution. Also, the checkpoint interval used by each federate is of importance in this perspective.

The overall time consumption and number of messages sent executing the federation, with and without fault-tolerance, will be measured and compared. This will be done to identify when the cost of having fault-tolerance will inhibit the simulation execution rather than make it more efficient, given a specific failure-rate of the federates.

It should also be noted that the effectiveness of the fault-tolerance mechanism presented here is clearly coupled with the mutual relations existing between federates. During migration, federates that are joined to the federation can still progress their execution. When the migrated federate joins, after being deployed in a new host environment, chances are that is has lagged behind the others, and thus it may start to produce messages in their past. However, this depends on the nature of the migrated federate. A federate publishing data of concern to a large audience will of course affect the effectiveness of the execution more than a federate producing data of less interest. In our test federation the mutual relation between the federates is organized in a fully connected network, and messages send to and from federates are completely randomized. In the future it will be of interest to test other federation topologies to investigate how the relations between federates influence the performance of our fault-tolerant approach.

REFERENCES

Berchtold, C., and M. Hezel. 2001. An architecture for fault-tolerant HLA-based simulation. In *Proceedings of the 15th European Simulation Multiconference*, 616-620. Prague, Czech Republic.

Bononi, L., G. D'Angelo, and L. Donatiello. 2003. HLA-based adaptive distributed simulation of wireless mobile systems. In *Proceedings of the 17th Workshop on Parallel and Distributed Simulation*, 40-49. San Diego, California.

Cai, W., S. Turner, and H. Zhao. 2002. A load management system for running HLA-based distributed simulations over the grid. In *Proceedings of the 6th IEEE International Workshop on Distributed Simulation and Real-Time Applications*, 7-14. Fort Worth, Texas.

Damani, O. P., and V. K. Garg. 1998. Fault-tolerant distributed simulation. In *Proceedings of the 12th Workshop on Parallel and Distributed Simulation*, 38-45. Alberta, Canada.

Eklöf, M., M. Sparf, and F. Moradi. 2004. Peer-to-peer-based resource management in support of HLA-based simulations. *Simulation* 80: 181-190.

Eklöf, M., J. Ulriksson, and F. Moradi. 2003. NetSim: An environment for network based modeling and simulation. In *Proceedings of the NATO RTO Symposium on C3I and M&S Interoperability*. Antalya, Turkey.

Huang, J., M. Tung, K. Wang, L. Hui, M. Lee, J. Wu, and S. Wai. 2003. Smart time management: The unified time management mechanism. In *Proceedings of the*

2003 European Simulation Interoperability Workshop. Stockholm, Sweden.

Jefferson, D. 1985. Virtual time. *ACM Transactions on Programming Languages and Systems* 7: 404-425.

Kiesling, T. 2003. Fault-tolerant distributed simulation: A position paper [online]. Available via http://fakinf.informatik.unibw-muenchen.de/~tkiesling/documents/ftds-position-paper.pdf [accessed March 21, 2005].

Lüthi, J., and S. Großmann. 2001. The resource sharing system: Dynamic federate mapping for HLA-based distributed simulation. In *Proceedings of the 15th Workshop on Parallel and Distributed Simulation*, 91-98. Lake Arrowhead, California.

Lüthi, J., and C. Berchtold. 2000. Concepts for dependable distributed discrete event simulation. In *Proceedings of the 14th International European Simulation Multi-Conference*, 59-66. Ghent, Belgium.

McBride, B. 2002. Jena: A semantic web toolkit. *IEEE Internet Computing* 6: 55-59.

McGuniess, D. L., and F. van Harmelen. 2004. OWL web ontology language overview [online]. Available via http://www.w3.org/TR/owl-features/ [accessed March 21, 2005].

Möller, B., M. Karlsson, and B. Löfstrand. 2005. Developing fault tolerant federations using HLA evolved. In *Proceedings of the 2005 Spring Simulation Interoperability Workshop*. San Diego, California.

Saleem, U. 2004. Developing java web services with AXIS [online]. Available via http://www.developer.com/java/web/article.php/3443951 [accessed March 21, 2005].

Tan, G., A. Persson, and R. Ayani. 2004. HLA federate migration. In *Proceedings of the 38th Annual Simulation Symposium*, 243-250. San Diego, California.

Vardanega, F., and C. Maziero. 2001. A generic rollback manager for optimistic HLA simulations. In *Proceedings of the 4th IEEE International Workshop on Distributed Simulation and Real-Time Applications*, 79-85. San Francisco, California.

Wang, X., S. J. Turner, M. Y. H. Low, and B. P. Gan. 2004. Optimistic synchronization in HLA based distributed simulation. In *Proceedings of the 18th Workshop on Parallel and Distributed Simulation*, 123-130. Kufstein, Austria.

Yan, H., Y. Zhang, G. Sun, and L. Zhong. 2003. Research on time warp mechanism in HLA. In *Proceedings of the 2nd International Conference on Machine Learning and Cybernetics*, 1092-1095, Xi-an, China.

AUTHOR BIOGRAPHIES

MARTIN EKLÖF is a research assistant at the Swedish Defence Research Agency (FOI), Department of Systems Modeling. He holds a Master of Science in Physical Geography from Lund University, Sweden, and is currently pursuing his PhD at the Royal Institute of Technology. His research interests include distributed simulations, especially infrastructures for composition and fault-tolerant execution of distributed simulations. His e-mail address is martin.eklof@foi.se.

RASSUL AYANI is Professor of Computer Science at the Department of Microelectronics and Information Technology, Royal Institute of Technology (KTH), Stockholm, Sweden. He received his BS degree from University of Technology in Vienna (Austria), his MS from University of Stockholm and his PhD from KTH in Stockholm. His current research interests are in distributed systems, performance analysis and distributed simulation. Dr. Ayani has served as program chair and program committee member at numerous international conferences. He is an associate editor of the ACM Transactions on Modelling and Computer Simulation (TOMACS). His e-mail address is rassul@imit.kth.se and his Web address is www.imit.kth.se/~rassul.

FARSHAD MORADI is a senior research officer and head of the Department of Systems Modelling at the Swedish Defence Research Agency (FOI). He has been working on modelling and simulation for the past seven years and has led projects at FOI since August 2000. He holds a Master of Science in Computer Science and Engineering from Chalmers University of Technology, Gothenburg, Sweden, and is pursuing his PhD at the Royal Institute of Technology (KTH). His research interests are in the area of Distributed and Web-based Simulations, Computer Generated Forces and Information Operations and Warfare. His e-mail address is farshad.moradi@foi.se.